Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2005-11

ACTIVITY SET LIST REPRESENTATION FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

by

Khaled Moumene
Jacques A. Ferland

April 2005

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

PUBLICATION
CRT-2005-11

ACTIVITY SET LIST REPRESENTATION FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

by

Khaled Moumene[1]

Jacques A. Ferland [1,2]

The research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), grant OGP0008312.

[1] Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 (moumenek@iro.umontreal.ca)

[2] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 (ferland@iro.umontreal.ca)

Département d'informatique et de recherche opérationnelle - Publication #1223

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2005

105-05213/

Abstract

This paper describes a new representation for the solutions of the resource-constrained project scheduling problem (RCPSP) named Activity Set List. The most efficient heuristics for the problem use the activity list representation and the serial SGS method to construct the corresponding solution (schedule). The activity list may include a search space of representations much larger then the space of schedules because the same schedule can correspond to many different activity list representations. We show how the activity set list representation can considerably reduce the search space and how to move more efficiently through it. We also show that this new representation never excludes the optimal solution and that it has many interesting properties.

Computational experiments show the efficiency of the activity set list. The performance of one of the best heuristics for the problem has been improved by introducing the new representation. The activity set list representation may be used to construct a new category of more efficient solution procedures for the problem.

Keywords: project scheduling, resource-constrained project scheduling, activity list representation, activity set list representation, heuristics and metaheuristics.

Résumé

Cet article décrit une nouvelle représentation des solutions du problème d'ordonnancement des activités sous contraintes de ressources nommée "Séquence d'ensembles d'activités". Les approches heuristiques les plus efficaces pour le problème utilisent la représentation en liste des activités et la méthode Serial SGS pour construire la solution correspondante (planning). L'espace de recherche contenant les représentations en liste des activités peut être beaucoup plus grand que celui des plannings car un même planning peut correspondre à plusieurs représentations en liste des activités différentes. Nous montrons comment les séquences d'ensembles d'activités peuvent considérablement réduire l'espace de recherche et comment il est possible d'y faire des déplacements plus efficaces. Nous montrons également que cette nouvelle représentation n'exclut jamais la solution optimale du problème et qu'elle possède de nombreuses propriétés intéressantes.

Les calculs expérimentaux que nous avons effectués montrent l'efficacité des séquences d'ensembles d'activités. Les performances d'une des meilleures heuristiques pour le problème ont été améliorées après l'introduction de la nouvelle représentation. Les séquences d'ensembles d'activités peuvent être utilisées pour construire une nouvelle catégorie de méthodes de résolution plus efficaces pour le problème.

Mots clés : problème d'ordonnancement, activités sous contraintes de ressources, représentation en liste des activités, séquence d'ensembles d'activités, heuristiques et métaheuristiques.

1 Introduction

In a Resource-Constrained Project Scheduling Problem (RCPSP), we consider a project including a set $I = \{1, 2, \cdots, N\}$ of N activities to be scheduled under resource and precedence constraints. The time required to complete an activity i is specified in terms of an integer number d_i of periods. Note that 1 and N are used as the starting and ending activities of the project, respectively, and their duration $d_1 = d_N = 0$. It is also assumed that once initiated, any activity is completed without interruption.

On the one hand, the precedence constraints are induced by technological requirements to impose that any activity i must be scheduled for execution after the completion of all its immediate predecessors included in a set P_i. On the other hand, each activity i requires r_{ik} units of resource $k \in R = \{1, 2, \cdots, K\}$ during each period of its execution. The resource constraints are specified to limit the number of units of each resource $k \in R$ used to the number A_{kt} of units available in each period t.

The (RCPSP) is to determine a starting period for each activity in order to minimize the total duration (makespan) of the project while satisfying the precedence and resource constraints. Note that this problem is denoted as $m/cpm/c_{max}$ or $PS/prec/C_{max}$ referring to Herroelen et al. [14] and Brucker et al.[7] classifications, respectively.

Since this problem is known to be a NP-Hard [3], solving problems having more than 60 activities and 4 different resources with an exact method is very much time consuming. Several authors have proposed different solution procedures for the (RCPSP). They can be classified into three categories. The exact methods [9, 12, 23, 24, 27, 28] are, essentially, implementation of the branch-and-bound to solve small problems having less than 60 activities. Using heuristic procedures is an alternative to deal with larger problems [4, 9, 11, 17, 19, 20, 29]. These methods are serial or parallel schedule generation schemes. The third category includes metaheuristics like tabu search [2, 25], simulated annealing [5, 6, 8] and genetic algorithm [1, 13, 16].

In this paper we introduce a new representation for the solutions of (RCPSP) called *activity set list*. The most efficient heuristics for the (RCPSP) are using activity list representation [18, 21] and the serial SGS method [21] to construct the corresponding solution (schedule). The activity list may induce a search space of representations much larger then the space of schedules because the same schedule can correspond to many different activity list representations. We show how the activity set

list representation can considerably reduce the search space and how to move more efficiently through it. We also show that this new representation never exclude the optimal solution and that it has many interesting properties.

Computational experiments show the efficiency of the activity set list. The performance of one of the best heuristics for the problem has been improved by introducing the new representation. The activity set list representation may be used to construct a new category of more efficient solution procedures for the problem.

The paper is organized as follows. In Section 2, we briefly describe the activity list representation and then define the activity set list representation. We introduce a powerful property of the new representation in Section 3 together with some results and procedures related to it. An example of numerical advantages of using the activity set list representation is presented in Section 4. In Section 5 we show how the new representation can be extended for the backward scheduling mode. Finally, a performance evaluation is made in Section 6 using a simulated annealing procedure very similar to Bouleimen and Lecocq method [6]. The numerical results for all instances in J30, J60, J90 and J120 of the PSPLIB [22] indicate the gain in efficiency when the new representation is used.

2 Activity Set List Representation

An essential feature of any heuristic solution procedure is the *encoding* or the *representation* of the solution. Note that many different procedures may use the same representation, but they differ in the way the solution representation are modified at each iteration. Several representations have been proposed [21], but one of the most commonly used is the *activity list representation* or *permutation based representation* [18, 21] where we use the following vector :

$$AL = [i_1, i_2, \cdots, i_N]$$

corresponding to a permutation of the activities satisfying the precedence constraints. Hence any activity i_j appears in AL after all its immediate predecessors P_{i_j}; i. e.,

$$P_{i_j} \subset \{i_1, i_2, \cdots, i_{j-1}\}$$

The activity list AL can be decoded into a unique schedule using the serial SGS method [21] as follows : activity i_1 is scheduled to start in the first period (i.e., at time 0). Activity i_j is the j^{th} activity scheduled as early as possible after the completion of all its predecessors in P_{i_j} according to the resource availability.

To illustrate the activity list representation, consider the example in Figure 1 where the project includes 10 activities requiring a unique type of resource. Assume that $A_{1t} = 6$ units of resource are available in each period t. With each activity-node i, we associate the pair (d_i, r_{i1}). Using the serial SGS method, the activity list,

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

is decoded into the unique feasible schedule illustrated in Figure 2. It is easy to verify that this schedule corresponds also to the other activity list,

$$[1, 2, 5, 3, 4, 6, 8, 7, 9, 10].$$

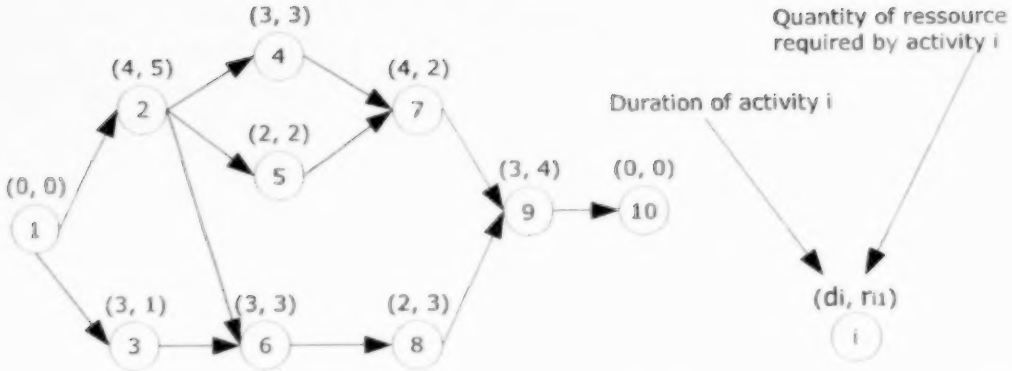

Figure 1: Example of (RCPSP)

This example illustrates the fact that the heuristic procedures working with this representation may generate several different activity lists corresponding to the same feasible schedule. Thus, these procedures search in a much larger space of activity lists than the space of feasible schedules. Now the following *activity set list representation* has the advantage of reducing the search space by representing several activity lists into a unique activity set list. To specify the activity set list representation we refer to the following definitions.

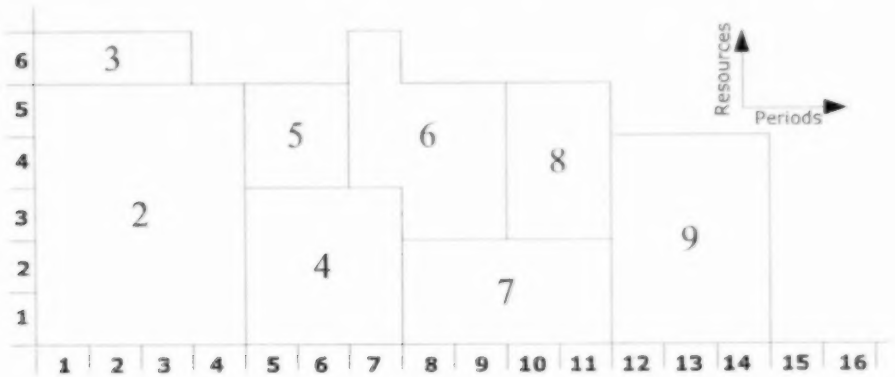

Figure 2: Example of a schedule

Definition 1 *An activity set list ASL is an ordered list of different non empty subsets of activities* $SA_1, SA_2, \cdots, SA_p$

$$ASL = [SA_1, SA_2, \cdots, SA_p]$$

such that

- *Each activity* $i \in I$ *belongs to a unique subset* SA_α, $\alpha \in \{1, 2, \cdots, p\}$ *(i.e., the set of subsets* $SA_1, SA_2, \cdots, SA_p$ *is a partition of* I*);*
- *If activity* $i \in SA_\alpha$, *then each of its predecessors is either in* SA_1 *or* SA_2 *or* $\cdots$ *or* SA_α *(i.e.,* $P_i \subset SA_1 \cup SA_2 \cup \cdots \cup SA_\alpha$*).*

It is easy to verify that the activity list AL is an activity set list ASL where each subset SA_j, $j = 1, 2, \cdots, N$, includes a *unique* activity.

Definition 2 *Consider a subset SA_α, an element of an activity set list ASL.*

- *os_α denotes an ordered list of activities of SA_α satisfying the precedence constraints (i.e., each activity $i \in SA_\alpha$ appears in os_α after its predecessors in SA_α $(P_i \cap SA_\alpha)$).*
- *OS_α is defined as the set of all ordered lists os_α that can be generated from SA_α.*

Definition 3 *An activity set list $ASL = |SA_1, SA_2, \cdots, SA_p|$ has the permutation property (or ASL is (PRMT)) if all vectors $|os_1, os_2, \cdots, os_p| \in OS_1 \times OS_2 \times \cdots \times OS_p$ are activity lists, and if they all correspond to the same unique schedule.*

To illustrate the definitions, consider the example in Figure 1. The activity set list

$$[\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}]$$

includes the subsets $SA_1 = \{1, 2, 3, 4, 5\}$ and $SA_2 = \{6, 7, 8, 9, 10\}$. Now, consider the ordered lists $os_1 = |1, 2, 5, 3, 4|$ and $os_2 = |6, 8, 7, 9, 10|$ satisfying the precedence constraints. Then the vector $|os_1, os_2| = |1, 2, 5, 3, 4, 6, 8, 7, 9, 10|$ is an activity list corresponding to the schedule of Figure 2. It is easy to verify that the combination of any ordered lists $|os_1, os_2| \in OS_1 \times OS_2$ in Table 1 is an activity list corresponding to the same unique schedule of Figure 2 (there are $8 \times 3 = 24$ possible combinations). Hence, the preceding activity set list is (PRMT). This example illustrates the advantage of working with activity set lists being (PRMT) to reduce the search space of a procedure.

Subsets	Sets of ordered lists
$SA_1 = \{1, 2, 3, 4, 5\}$	$OS_1 = \{\|1, 2, 3, 4, 5\|, \|1, 2, 3, 5, 4\|, \|1, 3, 2, 4, 5\|,$ $\|1, 3, 2, 5, 4\|, \|1, 2, 4, 5, 3\|, \|1, 2, 5, 4, 3\|,$ $\|1, 2, 4, 3, 5\|, \|1, 2, 5, 3, 4\|\}$
$SA_2 = \{6, 7, 8, 9, 10\}$	$OS_2 = \{\|7, 6, 8, 9, 10\|, \|6, 7, 8, 9, 10\|, \|6, 8, 7, 9, 10\|\}$

Table 1: Ordered lists generated from SA_1 and SA_2

Note that Definition 3 includes two conditions for an activity set list to be (PRMT). For instance, referring to the Figure 1, the activity set list

$$[\{1, 2, 3, 7, 8\}, \{4, 5, 6, 9, 10\}|$$

is not (PRMT) since the vector $[1, 2, 3, 7, 8, 4, 5, 6, 9, 10|$ does not verify the precedence constraint (is not an activity list) even if the ordered lists $os_1 = [1, 2, 3, 7, 8|$ and $os_2 = [4, 5, 6, 9, 10|$ do.

Moreover, all vectors induced by an activity set list can be activity lists but they may not correspond to the same unique schedule. Such an activity set list is not (PRMT). For instance, referring to 1, consider the following activity set list.

$$[\{1, 2, 3, 4, 5, 6\}\{7, 8, 9, 10\}|$$

We can easily verify that all vectors $|os_1, os_2| \in OS_1 \times OS_2$ are activity lists. Now, consider the activity lists

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10|$$
$$[1, 2, 3, 4, 6, 5, 7, 8, 9, 10|$$

from $OS_1 \times OS_2$. They correspond to different schedules that have makespan of 14 and 16, respectively.

3 Generating Activity Set List being (PRMT)

Next, we propose a procedure to generate activity set list being (PRMT). Starting with an activity list AL, select each activity sequentially. Initialize the first subset of the partition with the first activity in AL. Considering the starting period of the previous activities in AL, if the earliest starting period of the activity accounting only for the precedence constraints is the same if we account for both precedence and resource constraints, then the activity is included in the current subset. Otherwise, a new subset is initialized with this activity. The procedure is summarized in Figure 3.

- Let AL=$[i_1, i_2, \cdots, i_N]$ the activity list to initialize the procedure.
- For any $j = 1, 2, \cdots, N$, considering the starting period of the previous activities $i_1, i_2, \cdots, i_{j-1}$ in AL, denote :
  - $TP(i_j)$: The earliest starting period of i_j accounting only for the precedence constraints.
  - $TPR(i_j)$: The earliest starting period of i_j accounting for both the precedence and the resource constraints.

1. Initialize the first subset $SA_1 = \{i_1\}$.
2. $l = 1$.
3. **For** $k = 2$ **to** N **do***
   - **if** $(TP(i_k) = TPR(i_k))$ **then** $SA_l = SA_l \cup \{i_k\}$
   - **else**
     - $l = l + 1$
     - Initialize a new subset $SA_l = \{i_k\}$
4. The activity set list generated is $[SA_1, SA_2, \cdots, SA_l]$

Figure 3: The ConstructASLPRMT procedure

Note that the loop * is completed in $O(N)$. Furthermore, the values $TP(i_j)$ and $TPR(i_j)$ are evaluated in $O(cN^2)$, where c is a positive real number. Thus, ConstructASLPRMT is completed in polynomial time.

The procedure can induce less computational effort if AL has to be decoded. Indeed to generate the corresponding schedule, the activities in AL have to be selected sequentially and the values $TP(i_j)$ and $TPR(i_j)$ have to be evaluated. Hence, testing the if $TP(i_j)$ is equal to $TPR(i_j)$ for each selected activity i_j to determine its subset, is the only additional computation effort.

Referring to the example of Figure 1, it is easy to verify that starting with the activity list

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

the procedure ConstructASLPRMT generate the following activity set list being (PRMT)

$$[\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}]$$

The following result shows that ConstructASLPRMT generates an activity set list being (PRMT) when initiated with an activity list.

Theorem 1 *If initiated with an activity list AL, the ConstructASLPRMT generates an activity set list ASL being (PRMT)*

Proof

Referring to definitions 1, 2 and 3, for any order lists $os_1, os_2, \cdots, os_l \in OS_1 \times OS_2 \times \cdots \times OS_l$, denote the corresponding vector

$$V_0 : [os_1, os_2, \cdots, os_l]$$

We first show that V_0 satisfies the precedence constraints (i.e., V_0 is an activity list), and then we show that V_0 corresponds to the same schedule as AL.

1. V_0 is an activity list

   Consider an activity $i \in I$ and any of its predecessors $j \in P_i$. It follows that $i \in os_k$ for some $k \in \{1, 2, \cdots, l\}$.

   Since the procedure ConstructASLPRMT considers the activities sequentially in their order in AL, it follows that $j \in SA_1 \cup SA_2 \cup \cdots \cup SA_k$.

   - if $j \in SA_1 \cup SA_2 \cup \cdots \cup SA_{k-1}$, then j appears in either os_1 or os_2 or ... or os_{k-1}, thus the precedence constraint between i and j is satisfied.
   - if $j \in SA_k$, then j appears in os_k and by definition 2, it follows that the precedence constraint between i and j is also satisfied.

2. V_0 corresponds to the same schedule as AL

   We use a proof by induction to show that the activities in $SA_1, SA_2, \cdots, SA_k, k \leq l$, are scheduled to start with V_0 at the same period as with AL.

   - for $k = 1$: referring to AL, all activities $i \in SA_1$ are scheduled to start at their earliest starting period $TP(i)$. Now, since for any $os_1 \in OS_1$, the precedence constraints are satisfied, it follows that referring to V_0, all activities $i \in SA_1$ are also scheduled to start at their earliest starting period $TP(i)$.
   - Assume that the result is true for all activities in $SA_1 \cup SA_2 \cup \cdots \cup SA_k$. Now considering the starting period of the activities in $SA_1 \cup SA_2 \cup \cdots \cup SA_k$, we have to specify the starting period of the activities in SA_{k+1} according to the order $os_{k+1} \in OS_{k+1}$. Referring to the procedure ConstructASLPRMT and according to the order of these activities in AL, the first activity $i_1 \in SA_{k+1}$ is scheduled at period $TPR(i_1) > TP(i_1)$, and all the other activities $i \in SA_{k+1} - \{i_1\}$ are scheduled at period $TP(i)$. Now, considering the order os_{k+1} in V_0, two different cases are analyzed.

     - **Case 1 : i_1 is the first activity of SA_{k+1} to be scheduled according to the order os_{k+1}**

       All activities in $SA_1 \cup \cdots \cup SA_k$ are scheduled at the same period as with AL. Now, i_1 is the next activity to schedule. i_1 will be scheduled at the same period $TPR(i_1) > TP(i_1)$ as it was done with AL because i_1 was also the first activity to schedule with AL after the activities $SA_1 \cup \cdots \cup SA_k$. The rest of activities i in $SA_{k+1} - \{i_1\}$ will be scheduled at period $TP(i)$ because this was possible with AL.

– **Case 2 : i_1 is not the first activity of SA_{k+1} to be scheduled according to the order os_{k+1}**

In this case, denote

* SA^{bf} the set of activities in $SA_{k+1} - \{i_1\}$ appearing before i_1 in os_{k+1}.
* SA^{af} the set of activities in $SA_{k+1} - \{i_1\}$ appearing after i_1 in os_{k+1}.

For the activities in SA^{bf}, i_1 is not a predecessor. Hence, each $i \in SA^{bf}$ will be scheduled at period $TP(i)$ since it was possible to schedule it at this period according to the order in AL even if i_1 was scheduled at $TPR(i_1) > TP(i_1)$. Furthermore, it can not be scheduled earlier than $TP(i)$ because of the precedence constraints.

Activity i_1 will scheduled at period $TPR(i_1) > TP(i_1)$ since it was scheduled at this period according to the order in AL even with the activities $i \in SA^{bf}$ scheduled at period $TP(i)$.

Finally, each activity $i \in SA^{af}$ will be scheduled at the period $TP(i)$ since it was possible to schedule it at this period according to the order in AL even if i_1 is scheduled at $TPR(i_1) > TP(i_1)$ and the activities $j \in SA^{bf}$ are scheduled at $TP(j)$. Furthermore, the activities $i \in SA^{af}$ can not be scheduled earlier then $TP(i)$ because of the precedence constraints.

Then the schedule is the same as if the order in AL were used.

The proof that ConstructASLPRMT generates an activity set list ASL being (PRMT) is now completed.

The following result indicates a criterion to identify problems that can be solved with the CPM method.

Theorem 2 *If an activity set list generated with ConstructASLPRMT has a unique subset, then the optimal schedule can be obtained with the CPM method.*

Proof

Referring to the ConstructASLPRMT procedure, if all activities i belong to the same subset, then they are scheduled at their earliest period $TP(i)$ as if there were no resource constraints.

Kolisch [18], Sprecher et al. [26] have proved that if the (RCPSP) has an optimal schedule, there always exists an activity list corresponding to an optimal schedule when using the serial SGS method. It follows that if the ConstructASLPRMT procedure is initiated with this activity list, then it generates an activity set list (PRMT) corresponding to the same optimal schedule. This is summarized in the following result.

Theorem 3 *If the (RCPSP) has an optimal schedule, there always exists an activity set list (PRMT) corresponding to an optimal schedule.*

4 Numerical advantages of using Activity Set List Representation

Consider any local search technique where neighbour activity lists of AL are generated by selecting an activity and moving it to different other positions between its latest predecessor and its earliest successor in AL in order to satisfy the precedence constraints. Now, referring to the corresponding activity set list generated by the ConstructASLPRMT procedure initialized with AL,

$$ASL = [SA_1, SA_2, \cdots, SA_p]$$

it follows that AL corresponds to a vector

$$V_0 = [os_1, os_2, \cdots, os_p]$$

for some ordering $os_1, os_2, \cdots, os_p \in OS_1 \times OS_2 \times \cdots \times OS_p$. If the selected activity belongs to SA_j, then moving it to any other position within SA_j to maintain the precedence constraints satisfied, corresponds to generate a different $os_j \in OS_j$, but the (PRMT) property induces that this neighbour activity list correspond to the same schedule. It follows that this activity list is not an interesting neighbour to reduce the value of the objective function. Hence the effort can be reduced since the (PRMT) property allows to know beforehand that such an activity list should not be generated and decoded to evaluate its makespan.

To illustrate the reduction in computational effort, consider the first instance of J30 in the PSPLIB library [22] having 30 non dummy activities, and the activity list in Figure 4 generated with the MINSLK (Minimum Job Slack) priority rule [10]. Note that the first and the last activities 1 and 32, are dummy starting and ending activities.

[1, 4, 10, 16, 2, 9, 11, 21, 26, 5, 15, 6, 3, 13, 18, 20, 25, 7, 8, 27, 28, 31, 19, 29, 12, 14, 17, 22, 23, 24, 30, 32]

Figure 4: Activity list generated with MINSLK rule considering the first instance of J30

The corresponding activity set list generated with the ConstructASLPRMT procedure is illustrated in Figure 5.

{1, 4*, 10, 16, 2, 9, 11, 21, 26}{5, 15}{6}{3, 13}{18, 20, 25, 7, 8}{27, 28, 31, 19}
{29, 12, 14*}{17, 22, 23, 24, 30, 32}

Figure 5: Activity set list generated with the activity list in Figure 4

Suppose that the activity 9 is selected. It can be moved to any position after 4 and before 14 to maintain the precedence constraints satisfied. Hence 22 different neighbour activity lists can be generated according to these modifications. But moving activity 9 between activities 4 and 5 (i.e., within its subset SA_1) generates different order lists in OS_1 inducing 6 different activity lists corresponding to the same schedule. Hence, 27% of the 22 neighbour activity lists are useless since they correspond to the same schedule and do not have to be generated and decoded.

Additional tests have been completed with instances of the PSPLIB library as follows : for the first instance of each set J30, J60, J90 (including 30, 60 and 90 non dummy activities, respectively),

1. 100 different activity lists have been generated randomly according to the RAN priority rule (activities are selected randomly to construct valid activity lists);
2. For each activity list, the experimentation with an activity selected randomly is repeated 100 times;
3. For each selected activity, the percentage of useless neighbour activity lists has been evaluated.

Table 2 summarizes the average percentage of useless neighbour activity lists for each instance considered. Accordingly, nearly 28% of the neighbour activity lists are useless and do not have to be generated and decoded since they correspond to the same schedule as the original activity list. Hence, the computation effort to evaluate the neighbour solutions is reduced accordingly.

Problems	% of useless neighbours
J30	27
J60	29,75
J90	27,37

Table 2: Percentage of useless neighbours for problems J30, J60 and J90

Note that the restriction of moving an activity only at positions outside of its subset eliminates many useless movements but not necessarily all of them. Indeed, such a movement may also induce a new activity list corresponding to the same schedule.

5 Activity Set List Representation for the backward scheduling mode

So far, we have used the (forward) serial SGS introduced in Section 2 to decode an activity list into a schedule. Accordingly, we have introduced the notion of activity set list representation in reference with this scheduling mode. But an activity list can also be decoded into another schedule using a backward scheduling mode. This is also a serial SGS method where the last activity i_N is first scheduled. Activity i_{N-j+1} is the j^{th} activity scheduled as late as possible to be completed before starting its successors according to the resource availability.

Referring to the example of the Figure 1, the backward scheduling mode decodes the activity list

$$[1, 2, 5, 3, 6, 8, 4, 7, 9, 10]$$

into the schedule illustrated in Figure 6 which is different from the schedule, obtained with the forward mode, in Figure 7.

The notion of activity set list representation being (PRMT) can be extended in reference with the backward scheduling mode. To specify this new representation, we refer to the definitions 1 and 2 in Section 2, but we modify Definition 3 as follows :

Definition 4 *An Activity Set List $ASL = [SA_1, SA_2, \cdots, SA_p]$ has the backward permutation property (or ASL is (PRMTb)) if all vectors $[os_1, os_2, \cdots, os_p] \in OS_1 \times OS_2 \times \cdots \times OS_p$ are activity lists and if they all correspond to the same unique schedule when the backward scheduling mode is used.*

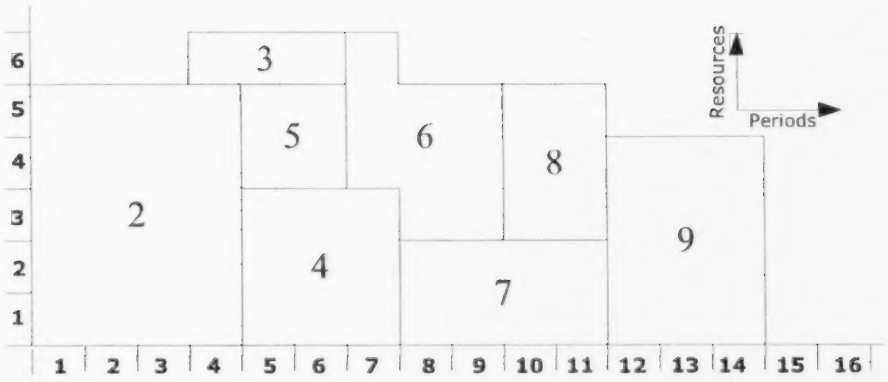

Figure 6: Schedule with the backward mode

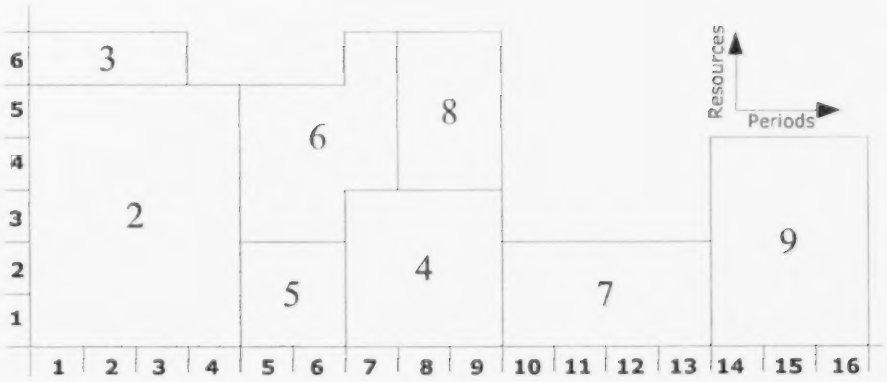

Figure 7: Schedule with the forward mode

Similarly, the ConstructASLPRMT procedure can be adapted into the ConstructASLPRMTb procedure to generate an ASL being (PRMTb) starting with an activity list $AL = [i_1, i_2, \cdots, i_N]$. It can be summarized as follows. Initialize the first subset SA_1 of the partition with i_N. Considering the starting period of the following activities $i_{j+1}, i_{j+2}, \cdots, i_N$, if the latest starting period $LP(i_j)$ of activity i_j accounting only for the precedence constraints is the same as $LPR(i_j)$ if we account for both precedence and resource constraints, i_j is included in the current subset. Otherwise, if $LPR(i_j) < LP(i_j)$, a new subset is initialized with i_j. A proof, similar to that of Theorem 1, can be used to show that the activity set list, generated by ConstructASLPRMTb,

$$[SA_l, SA_{l-1}, \cdots, SA_1]$$

is (PRMTb).

If we initialize the ConstructASLPRMTb procedure with the activity list $[1, 2, 5, 3, 6, 8, 4, 7, 9, 10]$, it is easy to verify that the activity set list generated is

$$[\{1, 2, 5\}\{3, 6, 8, 4, 7, 9, 10\}]$$

Note that this activity set list (PRMTb) is different from the activity set list (PRMT)

$$[\{1, 2, 5, 3, 6, 8\}\{4, 7, 9, 10\}]$$

generated with ConstructASLPRMT initialized with the same activity list. Thus, associated with the same activity list, we can generate different schedules and different activity set list according to the scheduling mode used.

A similar result to Theorem 2 is valid for the backward scheduling mode. If the activity set list being (PRMTb) generated with ConstructASLPRMTb includes a unique subset, the optimal schedule can be obtained with the CPM method.

In [15], Klein introduces the notion of *bidirectional planning* mode where each activity is either scheduled with the forward or with the backward mode. An extension of the activity set list representation may be explored for this mode.

6 Performance evaluation

To illustrate the efficiency of the activity set list representation, we use a solution procedure based on the simulated annealing metheuristic. This procedure is closely related to the one proposed by Bouleimen and Lecocq [6] which is one of the three best meheuristic procedures for the (RCPSP).

6.1 A simulated annealing procedure using Activity Set List Representation

As in the Bouleimen and Lecocq procedure, we use the activity list representation and the search is performed on C cooling chains. In each chain, S steps are made, and in each step $s \in \{1, \cdots, S\}$, N_s neighbours are generated. The value of N_s is initialized at value N_0 at the beginning of each chain, and it is progressively increased from step to step according to the rule $N_{s+1} = N_s(1 + sh)$. The parameter h is used to modulate the step length. After the last chain is completed, the neighbourhood of the best activity list found is completely explored.

In our procedure, unlike Bouleimen and Lecocq, rather than moving randomly an activity to any position between the latest predecessor and the earliest successor to generate a neighbour, we restrict this movement to be outside the activity subset as described in Section 4. Furthermore, the stopping

criteria of our procedure is the number $NbPlg$ of schedules or neighbours generated. For given values of the parameters C, S and h, N_0 is calculated as the minimum value that induces a number of schedules greater than or equal to $NbPlg$ if all the chains and the steps are completed. If this number is greater than $NbPlg$, the procedure is stopped when $NbPlg$ schedules are generated. The initial temperature T_0 is also calculated such that a neighbour of the initial schedule can be accepted with the probability 0.01 if it induces a degradation of 20 % of the makespan. For the sake of completeness we summarize the solution procedure in Figure 8.

6.2 Numerical experiments

Four different procedures are implemented and tested :

- **Planner_ASL** : our main procedure described above where the neighbourhood is generated with the activity set list representation.
- **Planner_AL** : same as Planner_ASL but where Bouleimen and Lecocq neighbourhood is generated with the activity list representation.
- **Planner_ASL_B** : same as Planner_ASL but the schedules are obtained with the *backward* mode.
- **Planner_AL_B** : same as Planner_AL but the schedules are obtained with the *backward* mode.

The four procedures use the same basic approach summarized in the Figure 8. The difference between them is in the way the neighbours are generated (using AL or ASL) and the mode (forward or backward) with which the schedules are obtained. These procedures were coded in Borland C++ Builder 6.0 and implemented on a PC having a 1.6 Ghz Pentium processor and 256 Mo of RAM running under Microsoft Windows XP.

The following numerical results allow to evaluate the impact on the procedure performance of using the activity set list representation being (PRMT) (respectively (PRMTb)). Two different sets of tests are completed using the standard library PSPLIB of instances generated by ProGen developed by Kolisch et al. [22]. PSPLIB is available at http://www.bwl.uni-kiel.de/prod/psplib/ and contains the sets of instances J30, J60, J90 and J120 having 30, 60, 90 and 120 activities respectively. J30, J60 and J90 contains 480 instances each, and J120 contains 600 instances. The last update date of PSPLIB was done in May 2003.

Our sets of tests can be described as follows :

- **Test 1** : Planner_ASL, Planner_AL, Planner_ASL_B and Planner_AL_B are tested and compared on J30, J60, J90 and J120 with different combinations of the parameters values $C \in \{1, 2, 5\}, h \in \{0, 1, 2\}$ and $S \in \{5\}$. The number $NbPlg$ of schedules generated is fixed at 5000. These values for the parameters were also used in [6] for J30.
- **Test 2** : Planner_ASL and Planner_AL are tested and compared on J30, J60 and J90 with the following parameter values : $C = 1, h = 1$ and $S = 5$. This selection of parameters is motivated by the good results that they generate in Test 1. The number $NbPlg$ of schedules generated is now fixed at 1000 times the number of activities. This allows equitable process time to explore the solution space of each instance.

In all these tests, α takes the value 0.25 as suggested in [6].

1. Generate the initial activity list (or solution) x_0 with the (SPT) priority rule heuristic (the priority is given to the activity having the Shortest Process Time). The makespan of the corresponding schedule is $f(x_0)$. Set $x_{Best} = x_0$ and $f_{Best} = f(x_0)$ as the best solution and its corresponding makespan, respectively.

2. Set $x_{Current} = x_0$ and $f_{Current} = f(x_0)$ as the current solution and its corresponding makespan.

3. The computational parameters are :

   - N_0 : the number of the initial neighbours.
   - h : a coefficient used to modulate the step length.
   - T_0 : the initial temperature.
   - T : the current temperature.
   - α : a parameter, between 0 and 1, used to decrease the current temperature.
   - C : the number of chains.
   - S : the number of steps.
   - $NbPlg$: the maximum number of schedules (or solutions) to generate before stopping the procedure.

4. Read the parameters h, α, C, S and $NbPlg$.

5. Calculate N_0 as the minimum value that induces a number of schedules greater than or equal to $NbPlg$ if all the chains and all the steps are completed.

6. Calculate T_0 such that a neighbour of x_0 can be accepted with the probability 0.01 if it induces a degradation of 20 % of $f(x_0)$.

7. **For** C chains **do**

   - $T = T_0$
   - $N_s = N_0$
   - **For** $s = 1, ..., S$ **do**
     - $N_s = N_s(1 + hs)$
     - **For** N_s neighbours **do**
       - **If** the number of solutions generated so far is ($NbPlg$ - the number of neighbours of x_{Best}) **then go to** the step 8 of the procedure.
       - Generate a neighbour x_v of $x_{Current}$.
       - Let $f(x_v)$, the makespan of the schedule corresponding to x_v.
       - Let $\Delta = f(x_v) - f(x_{Current})$.
       - **If** $\Delta < 0$ **then**
         - $x_{Current} = x_v$ and $f_{Current} = f(x_v)$.
         - **If** $f(x_v) < f_{Best}$ **then** $x_{Best} = x_v$ and $f_{Best} = f(x_v)$.
         - **If** $f(x_v) =$ CPM value **then** stop.
       - **Else If** $e^{(-\Delta/T)} >$ a value chosen randomly between 0 and 1 **then**
         - $x_{Current} = x_v$ and $f_{Current} = f(x_v)$.
     - **End For** N_s
     - $T = \alpha T$
   - **End For** s

   End For C

8. Explore the neighbourhood of x_{Best} .

Figure 8: The global schema of the solution procedure

6.3 Numerical results

The results of Test 1 are summarized in Tables 3, 4, 5 and 6 for instances J30, J60, J90 and J120, respectively. Each cell of these tables includes four different values associated with the four procedures as follows :

Planner_ASL (Planner_AL)
Planner_ASL _B (Planner_AL_B)

These values correspond to the *average percentage of deviation* (Average Dev.) or the *maximum percentage of deviation* (Max. Dev.) over all instances for a given selection of the parameters C and h. Note that for each instance, the percentage of deviation is evaluated with respect to the optimal or best results given in the May 2003 version of PSPLIB. Furthermore, for each procedure we indicate in ***bold character*** the best average percentage of deviation for each set of instances.

	C=1		C=2		C=5	
	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.
h=0	0.19 (**0.21**)	4.76 (5.43)	0.17 (**0.21**)	4.41 (5.26)	0.19 (0.25)	4.69 (5.17)
	0.18 (0.22)	5.17 (5.45)	0.21 (**0.20**)	7.89 (5.26)	0.17 (0.21)	6.58 (3.95)
h=1	**0.16** (0.24)	5.17 (5.45)	0.21 (0.23)	5.17 (5.17)	0.25 (0.26)	5.17 (5.97)
	0.19 (0.22)	5.45 (5.26)	**0.16** (0.21)	5.17 (5.19)	0.27 (0.28)	7.84 (6.90)
h=2	0.30 (0.37)	5.26 (6.78)	0.30 (0.37)	5.26 (6.78)	0.30 (0.37)	5.26 (6.78)
	0.30 (0.35)	5.17 (6.58)	0.30 (0.35)	5.17 (6.58)	0.30 (0.35)	5.17 (6.58)

Table 3: Test 1 : Results for J30 with 5000 schedules generated

	C=1		C=2		C=5	
	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.
h=0	1.27 (1.37)	11.49 (10.34)	1.29 (1.40)	10.34 (10.20)	1.39 (1.49)	10.19 (10.34)
	1.25 (1.39)	9.76 (10.31)	1.26 (1.38)	9.38 (9.91)	1.43 (1.53)	9.30 (10.83)
h=1	**1.14** (**1.28**)	9.09 (10.74)	1.39 (1.39)	11.49 (10.34)	1.55 (1.59)	11.11 (12.12)
	1.23 (**1.28**)	10.74 (10.00)	1.39 (1.42)	10.71 (10.74)	1.61 (1.71)	10.74 (11.67)
h=2	1.64 (1.77)	12.05 (15.15)	1.64 (1.77)	12.50 (15.15)	1.64 (1.77)	12.50 (15.15)
	1.64 (1.67)	11.63 (11.61)	1.64 (1.67)	11.63 (11.61)	1.64 (1.67)	11.63 (11.61)

Table 4: Test 1 : Results for J60 with 5000 schedules generated

The results in Tables 3 to 6 indicate the superiority of the versions of the procedures using activity set list representation over those using the activity list representation as far as the average percentage of deviation is concerned. Indeed, in only one case, in Table 4, Planner_AL generates results with the same average percentage of deviation as Planner_ASL, and in only one case in Table 3, Planner_AL_B generates results with better average percentage of deviation then Planner_ALS_B (a difference of 0.01 % in average deviation). In the other 70 cases, Planner_ASL and Planner_ASL_B generate better results than Planner_AL and Planner_AL_B, respectively.

The results of Test 2 comparing Planner_ALS an Planner_AL for instances J30, J60 and J90 with a different stopping criterion ($NbPlg$ equals to 1000×30, 1000×60 and 1000×90 for J30, J60 and J90, respectively) are summarized in Table 7.

	C=1		C=2		C=5	
	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.
h=0	**2.05** (2.25)	14.68 (13.28)	2.14 (2.32)	17.20 (14.29)	2.33 (2.35)	15.93 (15.38)
	2.00 (**2.14**)	14.29 (16.50)	2.17 (2.25)	16.81 (12.90)	2.29 (2.44)	15.45 (17.27)
h=1	2.09 (**2.12**)	15.38 (15.05)	2.15 (2.22)	14.52 (12.90)	2.59 (2.62)	14.66 (16.13)
	1.99 (**2.14**)	13.45 (14.29)	2.19 (2.27)	15.97 (13.76)	2.66 (2.75)	15.97 (17.48)
h=2	2.57 (2.61)	16.43 (15.05)	2.57 (2.61)	16.43 (15.05)	2.57 (2.61)	16.43 (15.05)
	2.43 (2.56)	15.97 (17.27)	2.43 (2.56)	15.97 (17.27)	2.43 (2.56)	15.97 (17.27)

Table 5: Test 1 : Results for J90 with 5000 schedules generated

	C=1		C=2		C=5	
	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.	Average Dev.	Max. Dev.
h=0	7.90 (8.59)	26.04 (25.00)	7.67 (**8.47**)	23.53 (27.08)	8.07 (8.94)	23.53 (25.81)
	7.52 (7.98)	21.32 (20.35)	7.18 (7.65)	19.12 (19.57)	7.60 (8.02)	22.06 (21.32)
h=1	**7.63** (8.48)	25.00 (33.61)	7.91 (8.54)	25.21 (24.04)	8.43 (9.00)	23.53 (27.06)
	6.95 (**7.23**)	22.79 (25.74)	7.26 (7.72)	20.59 (22.11)	8.03 (8.27)	21.24 (21.28)
h=2	8.42 (9.04)	24.71 (27.06)	8.42 (9.04)	24.71 (27.06)	8.42 (9.04)	24.71 (27.06)
	7.41 (7.68)	23.53 (17.65)	7.41 (7.68)	23.53 (17.65)	7.41 (7.68)	23.53 (17.65)

Table 6: Test 1 : Results for J120 with 5000 schedules generated

	Deviation		Number of schedules		Comparison with best		Computational time (Sec)	
	Average	Max	Average	Improvement	Equal	Improvement	Average	Max
J30	0.07 (0.07)	6.67 (2.67)	1149.56 (1485.09)	22.59%	463 (460)	0.65 %	3.34 (3.33)	8 (9)
J60	0.49 (0.54)	5.75 (5.75)	5906.59 (6938.20)	14.87%	378 (369)	2.43 %	8.91 (8.69)	34 (33)
J90	0.80 (0.85)	6.25 (6.85)	10898.63 (11356.80)	4.03%	366 (365)	0.27 %	15.91 (15.91)	63 (63)

Table 7: Test 2 : Results comparing Planner_ASL and Planner_AL with a number of schedules generated equal to 1000 times the number of activities; each cell includes : result of Planner_ASL (result of Planner_AL)

In addition to the *average percentage of deviation* and *maximum percentage of deviation*, Table 7 includes also the *average number of schedules* generated by Planner_ALS and Planner_AL (in parentheses) before reaching the best solution, and the percentage of improvement of Planner_ALS over Planner_AL with regards to this number. Furthermore, comparing with the best solutions reported in the May 2003 version of PSPLIB, we indicate the number of instances where the solutions generated are as good, and the percentage of improvement of Planner_ALS over Planner_AL with regard to this number. Finally, the last columns of the table include the average and the maximum computational time (in sec) to generate *NbPlg* schedules or to reach the CPM value. In average, Planner_ASL needs slightly more computation time than Planner_AL (half a second in the worst case) but induces solutions of better quality. This difference is due to the additional effort to guarantee that an activity is moved *outside its subset*. The results in Table 7 also indicate the superiority of Planner_ALS over Planner_AL as far as the average of percentage of deviation (except for J30 where equality holds), as far as the number of schedules generated before reaching the best solution, and as far as the number of problems where the best solution is reached, are concerned.

In [6], Bouleimen and Lecocq compare several heuristic procedures to solve the (RCPSP). They report that for instances J30, their simulated annealing procedure generate the best average percentage of deviation (0.23 %) after the genetic procedure of Alcaraz and Maroto [1] having a percentage of (0.123 %). Referring to Table 3, the procedures Planner_ALS and Planner_ALS_B can reach a percentage of 0.16 % indicating that these procedures would rank second after Alcaraz and Maroto procedure. It is difficult to make a similar comparison for instances in J60, J90 and J120 since the percentage of deviation in [1] were evaluated referring to the best known solutions that have been improved in the 2003 version of PSPLIB.

7 Conclusion

We present a new Activity Set List representation for (RCPSP). The most efficient heuristics for the problem are based on the activity list representation. We show how the Activity Set List can be seen as a generalization of the activity list where the same Activity Set List can be associated with many different activity lists corresponding to a same unique schedule. This property allows the construction of a new category of heuristics that are searching in a reduced feasible set.

Several tests have been made to illustrate the efficiency of the Activity Set List representation using a procedure based on one of the best heuristics in the literature. We compare its performance when using activity list and when using activity set list. The results clearly indicate the gain in performance induced by our new representation.

Further investigations need to be completed to explore more completely the impact of this new representation on heuristic procedures.

References

[1] Alcaraz J., Maroto C., *A Robust Genetic Algorithm for Resource Allocation in Project Scheduling*, Annals of Operations Research (102) 2001, 83-109

[2] Baar T., Brucker P., Knust S., *Tabu-search algorithms for the resource-constrained project scheduling problem*, Metaheuristics : Advances and Trends in Local Search Paradigms for Optimisation, Kluwer1997, 1-18

[3] Blazewicz J., Lenstra J.K., A.H.G. and Rinnooy Kan, *Scheduling projects to resource constraints : Classification and complexity*, Discrete Applied Mathematics (5) 1983, 11-24

[4] Boctor F.F., *Some effcient multi-heuristic procedures for resource-constrained project scheduling*, European Journal of Operational Research (49) 1990, 3-13

[5] Boctor F.F., *Resource-constrained project scheduling by simulated annealing*, International Journal in Production Research (34) 1996, 2335-2351

[6] Bouleimen M., Lecocq H., *A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple version*, European Journal Of Operational Research (149) 2003, 268-281

[7] Brucker P., Drexl A., Mohring R., Neumann K., *Resource-constrained project scheduling : Notation, classification, models and methods*, European Journal of Operational Research (112) 1999, 3-41

[8] Cho J.-H., Kim Y.-D., *A simulated annealing algorithm for resource-constrained project scheduling problems*, Journal of the Operational Research Society (48) 1997, 735-744

[9] Davis E.W., Heidorn G.E., *An algorithm for optimal project scheduling under multiple resource constraints*, Management Science (27) 1971, B803-B816

[10] Davis E.W., Patterson J.H., *A comparison of heuristic and optimal solutions in resource-constrained project scheduling*, Management Science (21) 1975, 944-955

[11] Demeulemeester E., Harroelen W., *New benchmarking results for the resource constrained project scheduling problem*, Management Science (43) 1995, 1485-1492

[12] Demeulemeester E., Herroelen W., *A branch-and-bound procedure for multiple resource-constrained project scheduling problem*, Management Science (38) 1992, 1803-1818

[13] Hartmann S., *A competitive genetic algorithm for resource-constrained project scheduling*, Naval Research Logistics (456) 1998, 733-750

[14] Herroelen W., Reyck B., Demeulemeester E., *Resource-constrained project scheduling : A survey of recent developments*, Computers and Operations Research (4) 1998, 279-302

[15] Klein R., *Bidirectional planning : improving priority rule-based heuristics for scheduling resource-constrained projects*, European Journal of Operational Research (127) 2000, 619-638

[16] Kohlmorgen U., Schmeck H., Haase K., *Experiences with fine-grained parallel genetic algorithms*, Anals of Operations Research (90) 1999, 203-219

[17] Kolisch R., Drexel A., *Adaptative search for solving hard project scheduling problem*, Naval Research Logistics (43) 1996, 23-40

[18] Kolisch R., *Serial and Parallel resource-constrained project scheduling methods revisited : Theory and computation*, European Journal of Operational Research (90) 1996, 320-333

[19] Kolisch R., *Efficient priority rules for the resource-constrained project scheduling problem*, Journal of Operations Management (14) 1996, 179-192

[20] Kolisch R., Drexl A., *Adaptative search for solving hard project scheduling problems*, Naval Research Logistics (43) 1996, 23-40

[21] Kolisch R., Hartmann S., *Heuristic algorithms for solving resource-constrained project scheduling problem : Classification and computation analysis*, in : J. Weglarz (Ed.), Project Scheduling : Recent Models, Algorithms and Applications, Kluwer Academic Publisher, Boston 1999, 147-178

[22] Kolisch R., Sprecher A., Drexel A., *Characterization and generation of general class of resource-constrained project scheduling problems*, Management Science (41) 1995, 1693-1703

[23] Mingozi A., Maniezzo V., Ricciardelli S., Bianco L., *An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation*, Management Science (44) 1998, 714-729

[24] Patterson J.H., Sowinski R., Talbot F.B., Weglarz J., *An algorithm for a general class of precedence and resource constrained scheduling problems*, in : R. Sowinski, 17.J.Weglarz (Eds), Advances in project scheduling, Elsevier, Amsterdam 1989, 3-28

[25] Pinson E., Prins C., Rullier F., *Using tabu search for solving the resource-constrained project scheduling problem*, in : Proceedings of the 4th International Workshop on Project Management and Scheduling, Leuven, Belgium 1994, 102-106

[26] Sprecher A., Kolisch R., Drexel A., *Semi-Active, active and non-delay schedules for resource-constrained project scheduling problem*, European Journal of Operational Research (80) 1995, 94-102

[27] Stinson J.P., Davis E.W., Khumawala B.M., *Multiple resource-constrained scheduling using branch-and-bound*, AIIE Transactions (10) 1978, 252-259

[28] Talbot B., Patterson J.H., *An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problem*, Management Science (24) 1978, 1163-1174

[29] Thesen A., *Heuristic scheduling of activities under resource and precedence restrictions*, Management Science (23) 1976, 412-422